SPRING BEAN DEFINITION INHERITANCE

A bean definition can contain a lot of configuration information, including constructor arguments, property values, and container-specific information such as initialization method, static factory method name, and so on.

A child bean definition inherits configuration data from a parent definition. The child definition can override some values, or add others, as needed.

Spring Bean definition inheritance has nothing to do with Java class inheritance but inheritance concept is same. You can define a parent bean definition as a template and other child beans can inherit required configuration from the parent bean.

When you use XML-based configuration metadata, you indicate a child bean definition by using the **parent** attribute, specifying the parent bean as the value of this attribute.

Example:

Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Create Java classes *HelloWorld*, *HelloIndia* and *MainApp* under the *com.tutorialspoint* package.
4	Create Beans configuration file *Beans.xml* under the **src** folder.
5	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Following is the configuration file **Beans.xml** where we defined "helloWorld" bean which has two properties *message1* and *message2*. Next "helloIndia" bean has been defined as a child of "helloWorld" bean by using **parent** attribute. The child bean inherits *message2* property as is, and overrides *message1* property and introduces one more property *message3*.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean  >
       <property name="message1" value="Hello World!"/>
       <property name="message2" value="Hello Second World!"/>
   </bean>

   <bean
       parent="helloWorld">
```

```
      <property name="message1" value="Hello India!"/>
      <property name="message3" value="Namaste India!"/>
   </bean>

</beans>
```

Here is the content of **HelloWorld.java** file:

```
package com.tutorialspoint;

public class HelloWorld {
   private String message1;
   private String message2;

   public void setMessage1(String message){
      this.message1  = message;
   }

   public void setMessage2(String message){
      this.message2  = message;
   }

   public void getMessage1(){
      System.out.println("World Message1 : " + message1);
   }

   public void getMessage2(){
      System.out.println("World Message2 : " + message2);
   }
}
```

Here is the content of **HelloIndia.java** file:

```
package com.tutorialspoint;

public class HelloIndia {
   private String message1;
   private String message2;
   private String message3;

   public void setMessage1(String message){
      this.message1  = message;
   }

   public void setMessage2(String message){
      this.message2  = message;
   }

   public void setMessage3(String message){
      this.message3  = message;
   }

   public void getMessage1(){
      System.out.println("India Message1 : " + message1);
   }

   public void getMessage2(){
      System.out.println("India Message2 : " + message2);
   }

   public void getMessage3(){
      System.out.println("India Message3 : " + message3);
   }
}
```

Following is the content of the **MainApp.java** file:

```
package com.tutorialspoint;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

      objA.getMessage1();
      objA.getMessage2();

      HelloIndia objB = (HelloIndia) context.getBean("helloIndia");
      objB.getMessage1();
      objB.getMessage2();
      objB.getMessage3();
   }
}
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```
World Message1 : Hello World!
World Message2 : Hello Second World!
India Message1 : Hello India!
India Message2 : Hello Second World!
India Message3 : Namaste India!
```

If you observed here, we did not pass message2 while creating "helloIndia" bean, but it got passed because of Bean Definition Inheritance.

Bean Definition Template:

You can create a Bean definition template which can be used by other child bean definitions without putting much effort. While defining a Bean Definition Template, you should not specify **class** attribute and should specify **abstract** attribute with a value of **true** as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean >
       <property name="message1" value="Hello World!"/>
       <property name="message2" value="Hello Second World!"/>
       <property name="message3" value="Namaste India!"/>
   </bean>

   <bean
       parent="beanTeamplate">
       <property name="message1" value="Hello India!"/>
       <property name="message3" value="Namaste India!"/>
   </bean>

</beans>
```

The parent bean cannot be instantiated on its own because it is incomplete, and it is also explicitly marked as *abstract*. When a definition is abstract like this, it is usable only as a pure template bean definition that serves as a parent definition for child definitions.